**TESTONTWERP & -REALISATIE**

|  |  |
| --- | --- |
| **Naam** | Mehmet Emin Ozturk |
| **Studentnummer** | 20129947 |
| **Klas** | DH-SE-7 |

**OPDRACHT**

Mijn opdracht is het maken van een overzichts-applicatie voor een auto dealership. In de applicatie kunnen dealership kijken welke voertuigen ze hebben, welke ze hebben verkocht en welke dealers werken bij de dealership.

Zo krijgt de dealership meer overzicht.

**GITHUB EN TDD**

Maak je GitHub-repo Private en geef je docent toegang tot je repo:

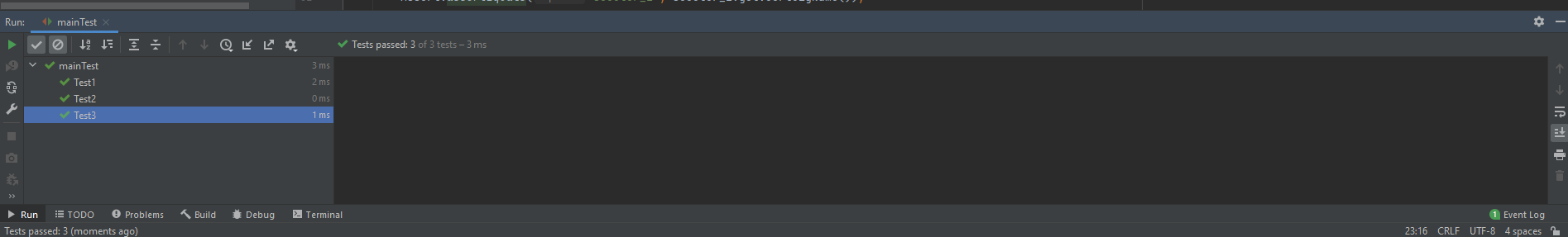
https://github.com/memooh/portfolioOpdracht2eersteJaar

Om aan te tonen dat je volgens TDD hebt gewerkt voor deze opdracht neem je hieronder een kopie van je netwerk over (zie de toelichting op het template voor instructie):



**GESLAAGDE TESTS**

Toon met een relevante screenshot aan dat alle onderstaande tests inderdaad zijn geslaagd:



**SAMENGESTELDE DECISION**

In mijn software neem ik een besluit op basis van de volgende voorwaarden (maximaal 3 te testen):

Voorwaarde A: Nieuwe dealer toevoegen aan een dealership

Voorwaarde B: Dealers aantonen van een dealership

Voorwaarde C: <voorwaarde C>

Het besluit is gebaseerd op de volgende combinatie van deze voorwaarden (bijv. D = A && (B || C)):

D = A && B

**CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D =** A && B) |
| 1 | 0 | <waarde> | 1 |
| 0 | 1 | <waarde> | 1 |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D =**  A && B |
| 0 | 0 | <waarde> | 0 |
| 0 | 1 | <waarde> | 1 |
| 1 | 0 | <waarde> | 1 |
| <waarde> | <waarde> | <waarde> | <waarde> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition/Decision Coverage de volgende test cases testen:

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = A && B** |
| 0 | 0 |  | 0 |
| 1 | 1 |  | 1 |
| <waarde> | <waarde> |  | <waarde> |
| <waarde> |  | <waarde> | <waarde> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**MODIFIED CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Modified Condition/Decision Coverage de volgende test cases testen:

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D =** A && B |
| 1/”Fabiolo” | 0/null | <waarde> | 1 |
| 1/”Fabiolo” | 1/”full\_list” | <waarde> | 1 |
| 0/”null” | 0/”null | <waarde> | 0 |
| 0/”null” | 1/”full\_list” | <waarde> | 1 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**MULTIPLE CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Multiple Condition Coverage de volgende test cases testen:

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| **0** | **0** | <waarde> | **0** |
| **0** | **1** | <waarde> | **1** |
| **1** | **0** |  | **1** |
| **1** | **1** | <waarde> | **1** |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
|  |  |  |  |

**MODIFIED CONDITION/DECISION COVERAGE (JUNIT)**

Ik heb de testcases voor Modified Condition/Decision Coverage omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

<De JUnit-code (o.b.v. MC/DC>

// Test of je dealers kan toevoegen aan een dealership.

@Test

public void Test1() {

Dealer dealer\_1 = new Dealer("test","test","dd@test.nl",18);

Dealership dealership\_1 = new Dealership("dealership\_1","locatie\_1");

dealership\_1.addDealer(dealer\_1);

// Dealer wordt toegevoegd, waardoor dealership 1 dealer heeft namelijk .get(0) en hierop wordt de test uitegevoerd.

Assert.assertEquals(dealer\_1, dealership\_1.getHasA().get(0));

}

Hiermee test ik de volgende methode (van het type boolean, waarin de waarde van D = <Decision> wordt bepaald):

class Dealership {

private String naam;

private String locatie;

private ArrayList<Dealer> hasA = new ArrayList<>();

public Dealership(String naam, String locatie) {

this.naam = naam;

this.locatie = locatie;

}

public void addDealer(Dealer dealer) {

hasA.add(dealer);

}

public void getDealers() {

for (Dealer m : hasA) {

System.out.println(m.getVoornaam() + " " + m.getAchternaam());

}

}

public ArrayList<Dealer> getHasA() {

return hasA;

}

public void dLocatie(){

System.out.println("Locatie is: " + this.getLocatie());

System.out.println("En de naam van deze franchise is " + this.getNaam());

}

public String getNaam() {

return naam;

}

public void setNaam(String naam) {

this.naam = naam;

}

public String getLocatie() {

return locatie;

}

public void setLocatie(String locatie) {

this.locatie = locatie;

}

}

|  |  |
| --- | --- |
|  |  |

**EQUIVALENTIEKLASSEN EN RANDWAARDEN**

Voor het nemen van een besluit in de software heb ik een domein dat bestaat uit de volgende equivalentieklassen (maximaal 3 te testen):

|  |  |  |
| --- | --- | --- |
|  | Logische weergave | Besluit |
| Equivalentieklasse 1 | 0 < Prijs == 1 | €34000,- |
| Equivalentieklasse 2 | 1 < Prijs == 2 | €18000,- |
| Equivalentieklasse 3 | #Prijs > 2 | €8000,- |

Relevante fysieke testgevallen:

|  |  |  |
| --- | --- | --- |
| Equivalentieklasse | Fysiek testgeval | Besluit |
| 0 < Prijs == 1 | 0 | Geen prijs |
| 1 < Prijs == 2 | 1 | 34000 |
| #Prijs > 2 | 2 | 8000 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**EQUIVALENTIEKLASEN EN RANDWAARDEN (JUNIT)**

Ik heb de testcases voor Equivalentieklassen en randwaarden omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

@Test  
public void getPrijs() {  
 Prijs voertuig1 = new Prijs(1);  
 Assert.*assertEquals*(13600, voertuig1.getPrijs(1), 0.01);  
}

Hiermee test ik de volgende methode (waarin de besluiten op basis van de equivalentieklassen worden bepaald):

class Prijs {

private Integer aantal;

public Prijs(Integer aantal) {

this.aantal = aantal;

}

public double getPrijs(int prijs) {

int dealpercentage = 0;

double total = 0.00;

if (prijs > 1000 && prijs <= 14000) {

dealpercentage = 20;

}

if (prijs > 0 && prijs<= 1000) {

dealpercentage = 40;

}

if (prijs > 14000 && prijs<= 100000) {

dealpercentage = 0;

}

if (voertuigNaam.equals("bmw".toUpperCase(LOCALE.ROOT)) && aantalDeuren == 2){

dealpercentage = 10;

} else if(voertuignaam.equals("audi".toUpperCase(LOCAL.ROOT)) aantalDeuren == 1){

dealpercentage = 20;

} else if (voertuignaam.equals("mercedes".toUpperCase(LOCAL.ROOT)) aantalDeuren == 4){

dealpercentage = 15;

} else{

dealpercentage = 0;

}

for (int i = 1; i <= this.aantal; i++) {

if (i == 2) {

total += 18000.00;

}

if (i == 1) {

total += 34000.00;

}

if (i > 2) {

total += 8000.00;

}

}

return total / 100 \* dealpercentage;

}

public Integer getAantal() {

return aantal;

}

public void setAantal(Integer aantal) {

this.aantal = aantal;

}

}

}

**PAIRWISE TESTING**

Voor Pairwise Testing maak ik gebruik van de volgende parameters (op basis waarvan een besluit wordt genomen; vereenvoudig tot een set van 4 parameters die ieder 3, 2, 2 en 2 mogelijkheden hebben (anders wordt handmatig testen met Pairwise Testing veel te ingewikkeld; vul een spatie in, als je een optie niet wilt gebruiken):

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Optie 1** | **Optie 2** | **Optie 3** |
| Prijs | 0 < Prijs <== 1000 | 1000 < Prijs == 14000 | 14001 < Prijs == 100000 |
| Voertuig | Audi | BMW | Mercedes |
| Aantal Deuren | 1 | 2 | 4 |
|  |  |  |  |

Na de tussenstappen in het proces heb ik de volgende testtabel met fysieke testgevallen opgesteld:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M**in prijs | **M**ax prijs | **Voertuig** | **Aantal Deuren** | **Decision** |
| 0 | 1000 | Audi | 1 | 40% korting op totaalprijs |
| 1000 | 14000 | BMW | 2 | 20% korting op totaalprijs |
| 14000 | 100000 | Mercedes | 4 | 0% korting |
|  |  |  |  |  |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**PAIRWISE TESTING (JUNIT)**

Ik heb de testcases voor Pairwise Testing omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

@Test  
public void getPrijs() {  
 Prijs voertuig1 = new Prijs(1);  
 Assert.*assertEquals*(13600, voertuig1.getPrijs(1), 0.01);  
}

Hiermee test ik de volgende methode (waarin het besluit op basis van de meegegeven parameters wordt bepaald):

class Prijs {

private Integer aantal;

public Prijs(Integer aantal) {

this.aantal = aantal;

}

public double getPrijs(int prijs) {

int dealpercentage = 0;

double total = 0.00;

if (prijs > 1000 && prijs <= 14000) {

dealpercentage = 20;

}

if (prijs > 0 && prijs<= 1000) {

dealpercentage = 40;

}

if (prijs > 14000 && prijs<= 100000) {

dealpercentage = 0;

}

if (voertuigNaam.equals("bmw".toUpperCase(LOCALE.ROOT)) && aantalDeuren == 2){

dealpercentage = 10;

} else if(voertuignaam.equals("audi".toUpperCase(LOCAL.ROOT)) aantalDeuren == 1){

dealpercentage = 20;

} else if (voertuignaam.equals("mercedes".toUpperCase(LOCAL.ROOT)) aantalDeuren == 4){

dealpercentage = 15;

} else{

dealpercentage = 0;

}

for (int i = 1; i <= this.aantal; i++) {

if (i == 2) {

total += 18000.00;

}

if (i == 1) {

total += 34000.00;

}

if (i > 2) {

total += 8000.00;

}

}

return total / 100 \* dealpercentage;

}

public Integer getAantal() {

return aantal;

}

public void setAantal(Integer aantal) {

this.aantal = aantal;

}

}

class Auto extends Voertuig {

private Integer kmStand;

private Integer aantalDeuren;

public Auto(String name, Integer kmStand, Integer aantalDeuren) {

super(name);

this.aantalDeuren = aantalDeuren;

this.kmStand = kmStand;

}

@Override

public void Opslag() {

System.out.println("Auto heeft merk "+ this.getVoertuigName());

System.out.println("Heeft als kilometerstand: " + this.getKmStand());

System.out.println("En heeft zoveel deuren: " + this.getAantalDeuren());

}

public Integer getKmStand() {

return kmStand;

}

public void setKmStand(Integer kmStand) {

this.kmStand = kmStand;

}

public Integer getAantalDeuren() {

return aantalDeuren;

}

public void setAantalDeuren(Integer aantalDeuren) {

this.aantalDeuren = aantalDeuren;

}

}